Innokentii Sennovskii (@Rumata888)

Multisignatures

whoami

- Currently work at Aztec Protocol as an Applied Cryptographer
- Interested in Applied Cryptography and Fuzzing
- Discovered Spectre Variant 3a

Talk agenda

- Math (sorry)
- Regular signatures
- MPCing regular signatures
- A bit more math (again, sorry)
- Aggregate signatures

Signature algorithms

- DSA (ElGamal)
- ECDSA
- RSA (PKCS#1v1.5, PSS)
- Schnorr
- etc.

Groups

A set G with a binary operation $*$, $\forall a, b \in \mathbb{G} \Rightarrow a * b \in \mathbb{G}$ and:

- Associativity ($\forall a, b, c \in \mathbb{G},\ (a * b) * c = a * (b * c)$)
- Identity element ($\exists\ e \in \mathbb{G} : \forall a \in \mathbb{G},\ a * e = e * a$)
- Inverse element ($\forall a \in \mathbb{G},\ \exists b \in \mathbb{G} : a * b = e = b * a$)

A group is called abelian if its group operation is commutative:

- $\forall a, b \in \mathbb{G},\ a * b = b * a$

Additive groups

- For example $\forall a, b \in \mathbb{Z}_5, c = a + b \; mod \; 5$
- Associativity - obvious
- Identity - 0
- Inverse - $\forall a \in \mathbb{G}, \; a + (5 - a) \; = 0 \; mod \; 5$

Scalar multiplication/exponentiation

We repeat the same operation on the element k times:

$$\forall a \in \mathbb{G}(+),\ \underbrace{a + +a \ldots + a}_{k} = k \cdot a$$

Groups in cryptography

We want all of this at ounce:

- Cyclic groups (generated by a single element)
- Efficient scalar multiplication/exponentiation
- Hard discrete logarithm (inverse of the previous operation)

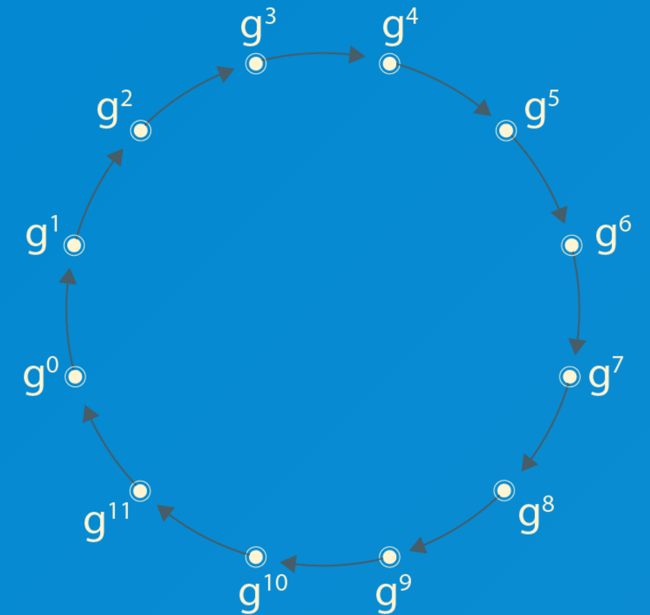
Additive groups modulo N don't cut it.

Euler's totient

- $N = p_1^{k_1} \cdot \ldots \cdot p_l^{k_l}$
- $\varphi(N) = \prod_{i=1}^{l}(p_i^{k_i - 1} \cdot (p_i - 1))$
- $\varphi(p) = p - 1$
- $\varphi(pq) = (p - 1)(q - 1)$

Multiplicative groups

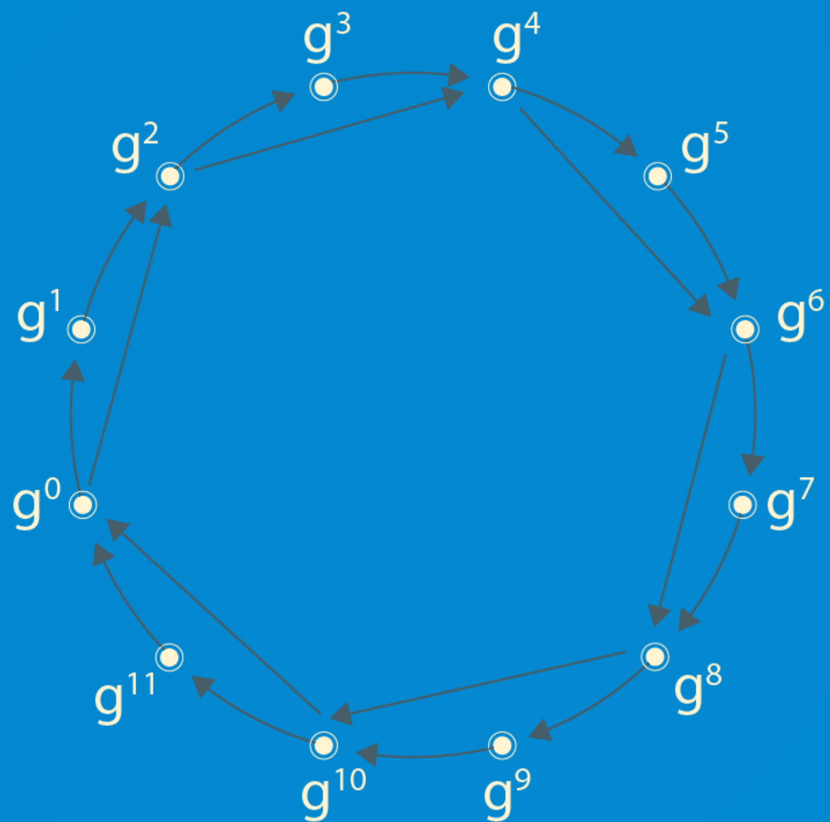
- $\mathbb{Z}_N^*$
- $|\mathbb{Z}_N^*| = \varphi(N)$
- if $N = p$ elements of $\mathbb{Z}_p^*$ are in $[1, p-1]$

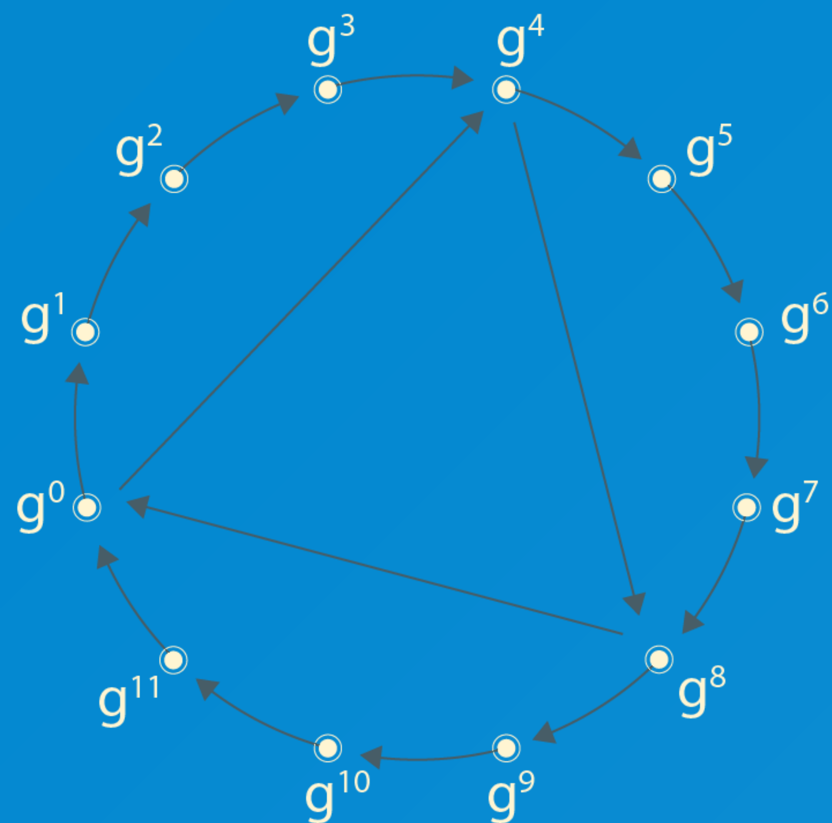


RSA

- $N = p * q$
- $\varphi(N) = (p - 1)(q - 1)$
- $e * d = 1\ mod\ (p - 1)(q - 1)$
- $Enc(m) = m^e\ mod\ N$
- $Dec(c) = c^d\ mod\ N$
- $Dec(Enc(m)) = (m^e)^d\ mod\ N = m^{ed}\ mod\ N$

Subgroups



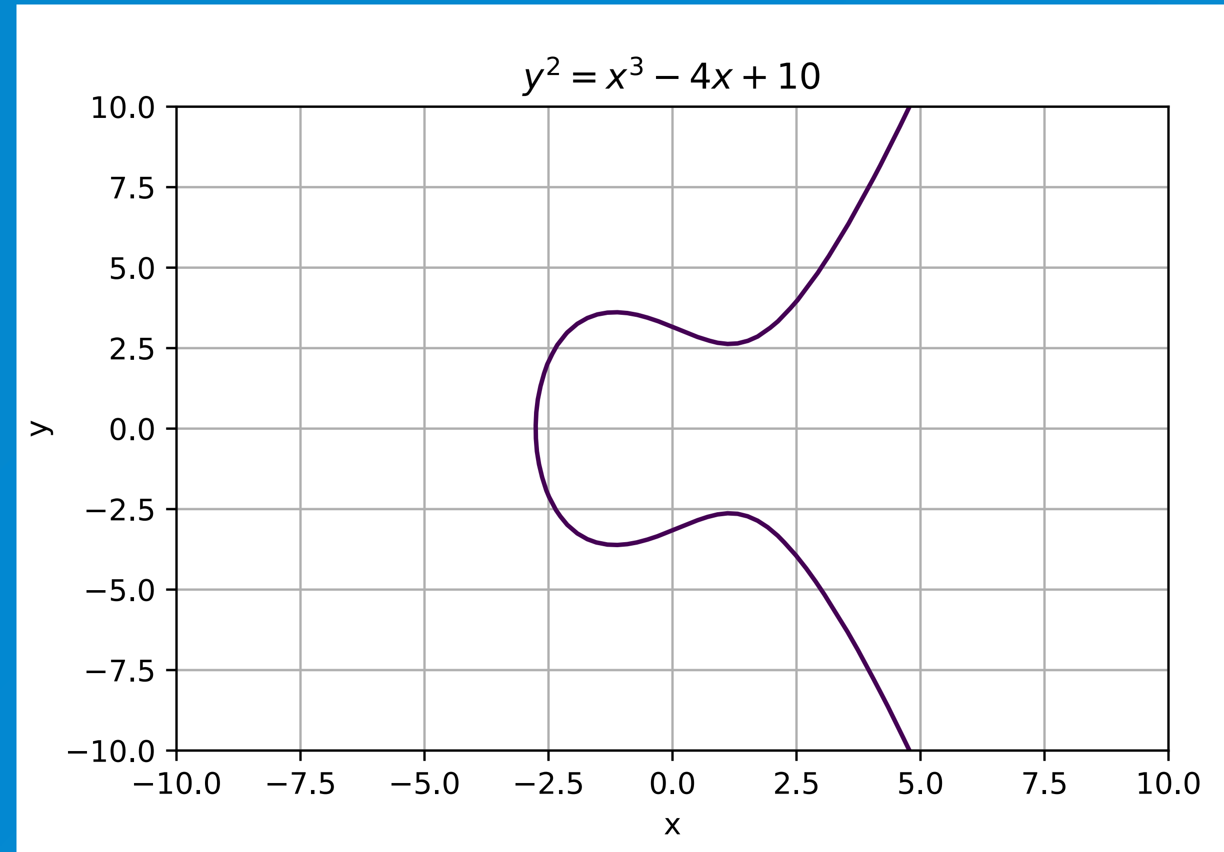


Secure multiplicative groups

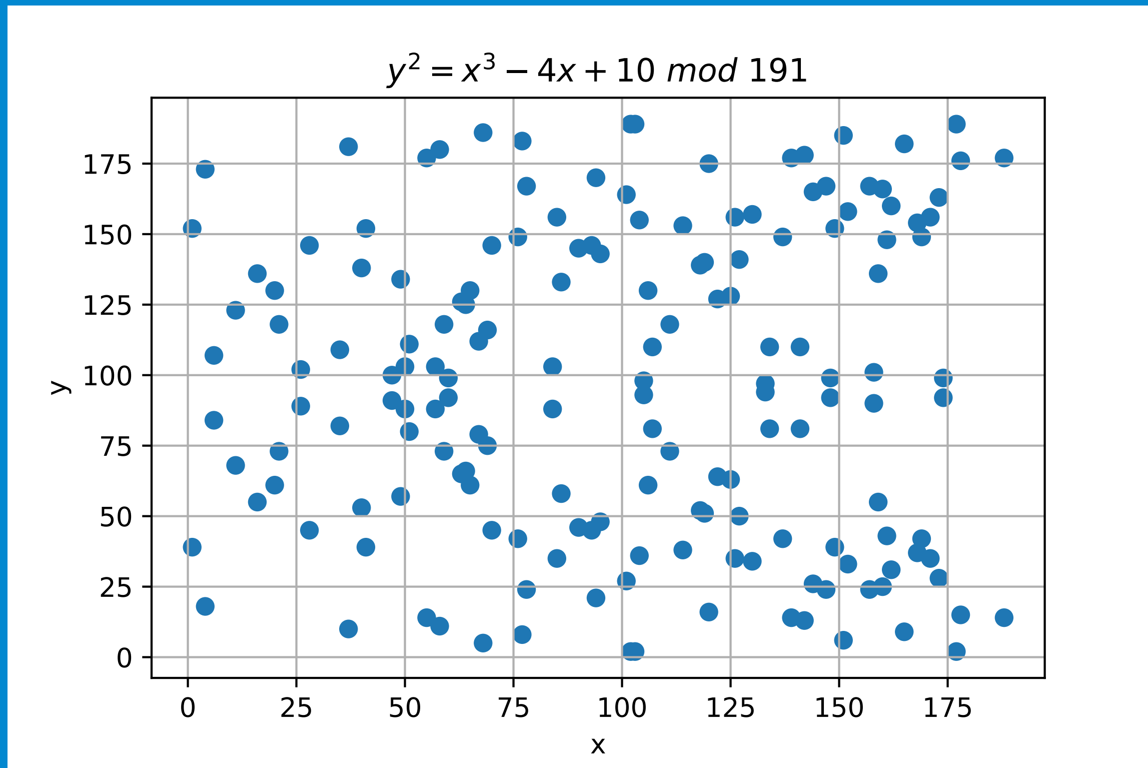
- Prime $p = 2q + 1$ (q is a prime, too)
- $|\mathbb{Z}_p^*| = p - 1 = 2q$
- Generator $g \in \mathbb{Z}_p : g \neq 0, 1, p - 1$
- $g^{p-1} = 1 \ mod \ p$

Elliptic curves

- $y^2 = x^3 + ax + b$ - short Weierstrass form

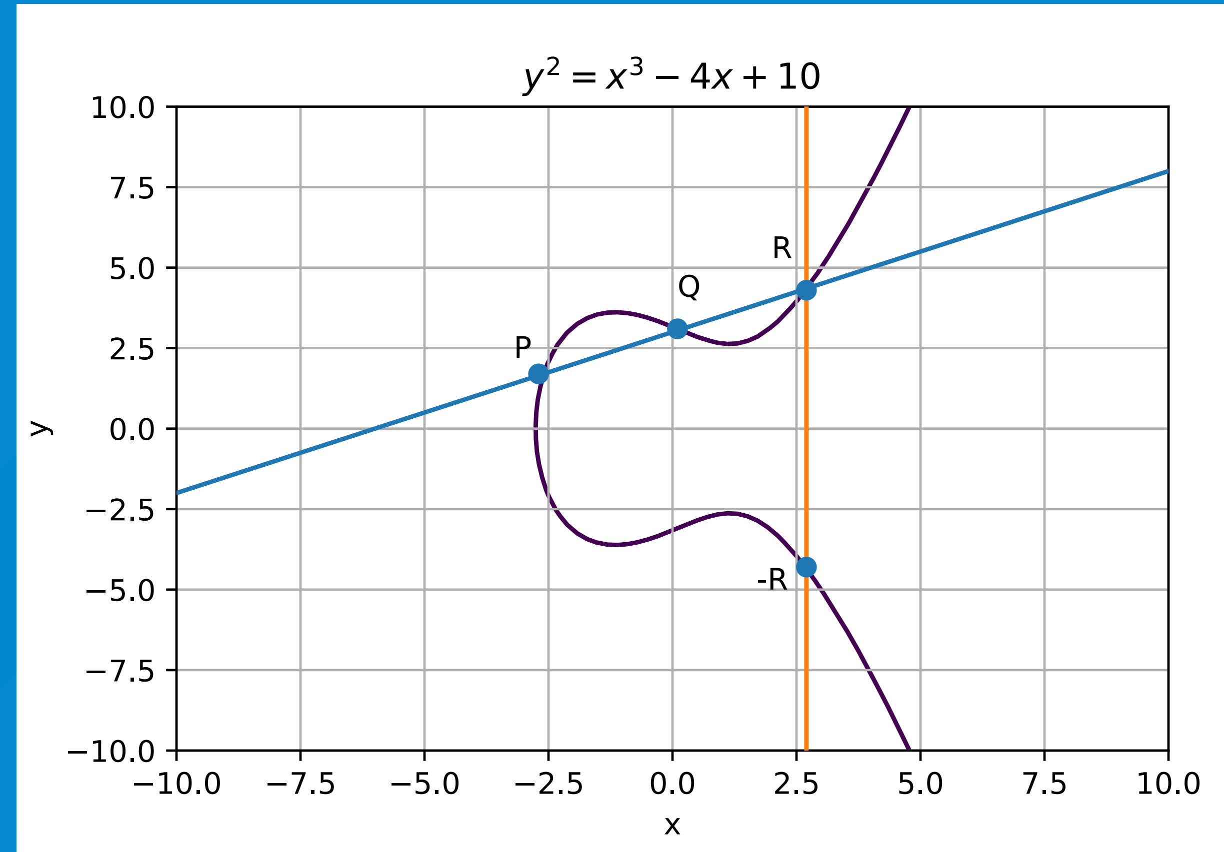


EC in Prime Field



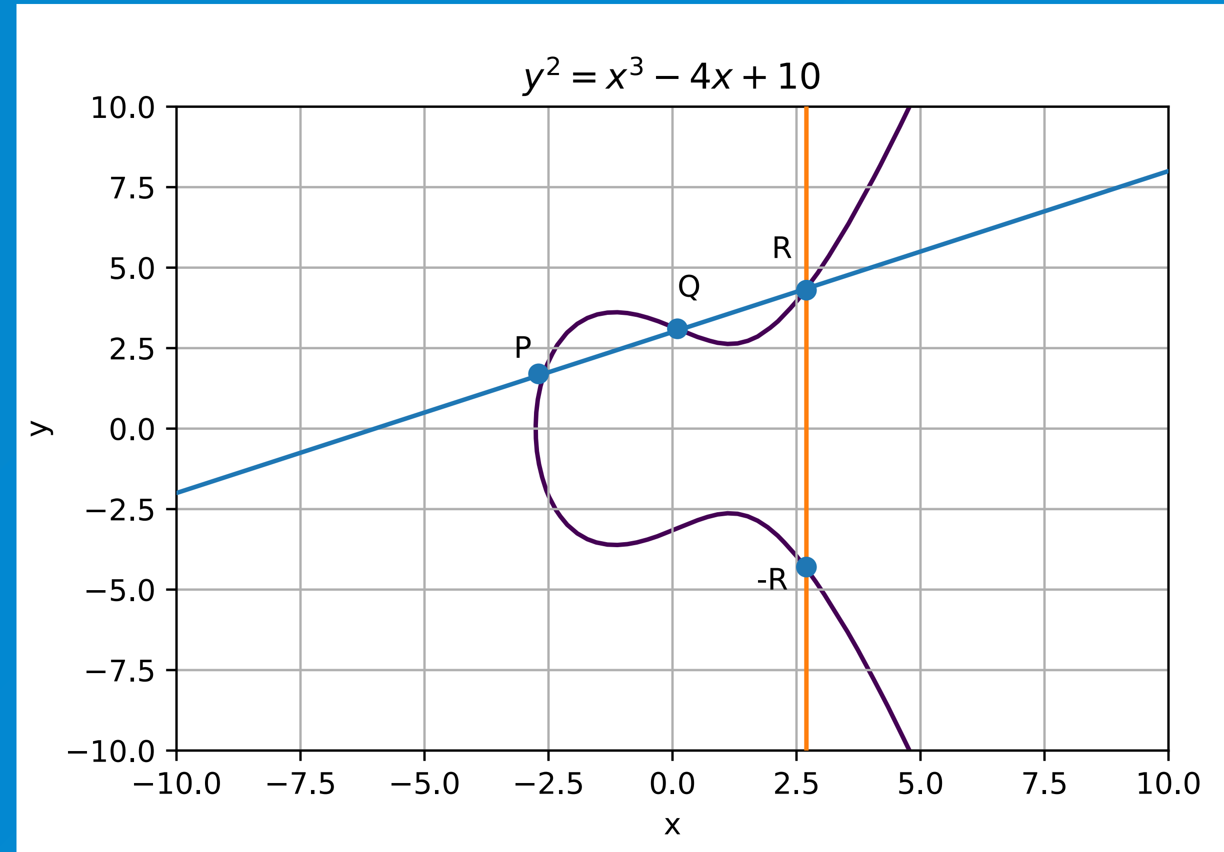
EC Points Group

- 0 - point at infinity
- $P + Q + R = 0$
- $-R(x, y) = R(x, -y)$



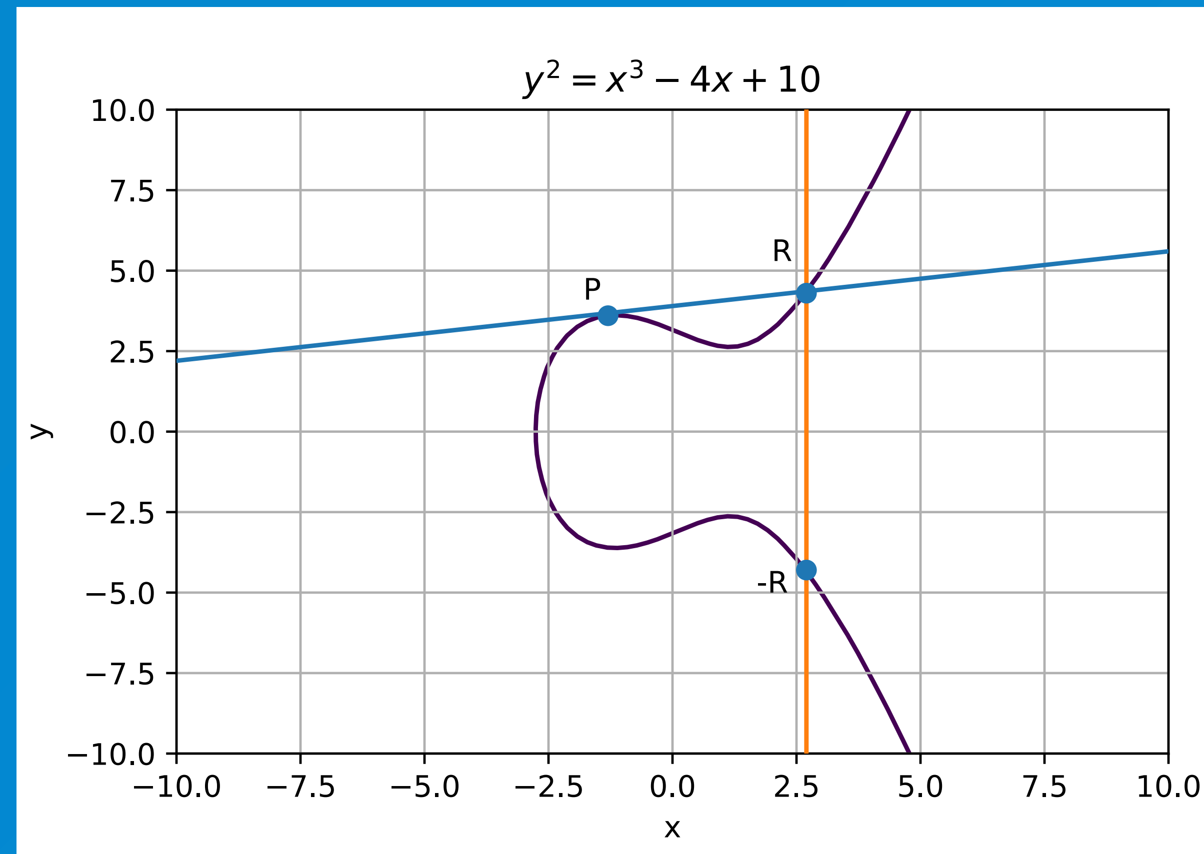
$P \neq Q$

- $P(x_1, y_1), Q(x_2, y_2)$
- $M = \frac{y_2 - y_1}{x_2 - x_1}$
- $c = y_2 - M \cdot x_2$
- $R_x = M^2 - x_1 - x_2$
- $R_y = M \cdot R_x + c$
- $P + Q = (R_x, -R_y)$



$P = Q$

- $P(x, y)$
- $M = \frac{3*x^2+a}{2y}$
- $c = y - M \cdot x$
- $R_x = M^2 - 2x$
- $R_y = M \cdot R_x + c$
- $P + Q = (R_x, -R_y)$



Scalar multiplication in EC Points Group

- $P \in E(\mathbb{F})$
- $Q = \underbrace{P + \ldots + P}_{k} = k \cdot P$

Computing scalar multiplication (double-add)

```
result = 0
for k_i in k: # i:255 -> 0
    result = Double(result)
    if (k_i):
        result+=P
```

Benefits of EC Group Crypto

- Considered a black-box group
- Z_p^* speed-ups do not apply
- 256 bits in Curve order ~ 3072 bits in Z_p^*
- It's possible to create groups of any given order
- Quite efficient algorithms for point computation exist
- The impact of smaller modulus outweight the number of operations

Computation comparison

- $\underbrace{k_0 \ldots k_{63}}_{limb_0}\underbrace{k_{64} \ldots k_{127}}_{limb_1}\underbrace{k_{128} \ldots k_{191}}_{limb_2}\underbrace{k_{192} \ldots k_{255}}_{limb_3}$
- Multiplication ~ 16 limb multiplications
- 3072-bits ~ 48 limbs on x64
- Multiplication ~ 2304 limb multiplications

Computation comparison

- 3072 double-add $1.5 * 3072$ operations on average
- 256 double-add $1.5 * 256$ on average
- $\frac{1.5*3072*2304}{1.5*256*16} = 1728$

One doubling/addition operation on EC has to consist of 1728 (roughly) multiplication operations to be as slow as $\mathbb{Z}_p^*$ of the same security level

Signatures

RSA

- $N = p * q$
- $\varphi(N) = (p-1)(q-1)$
- $e * d = 1 \ mod \ (p-1)(q-1)$
- $Sign(m) = Pad(H(m))^d \ mod \ N$
- $Verify(s) : Unpad(s^e \ mod \ N) = H(m)$

RSA problems

- p, q have to be prime and secret
- d has to be secret
- d can't be small
- Tons of bugs in padding in the past
- Just read "Twenty Years of Attacks on the RSA Cryptosystem" by Dan Boneh

DSA

- q - largest prime dividing $p-1$
- $h = \frac{p-1}{q}$ - cofactor
- $g' \in \mathbb{Z}_p^* => g = g'^h \mod_p \neq 1$ (usually $g = 2$)
- Public parameters: (p, q, g)

DSA key generation

- Pick private key $x \in \{1, q - 1\}$
- Public key $y = g^x \mod p$

Signing

- Pick random "nonce" $k \in \{1, q-1\}$
- $r = g^k \mod p$
- $s = \frac{H(m)+rx}{k} mod\ q$
- Output (r, s)

Verification

- Check $r, s \in \{1, p-1\}$
- $u_1 = \frac{H(m)}{s} \ mod \ q$
- $u_2 = \frac{r}{s} \ mod \ q$
- Check $g^{u_1} y^{u_2} = r \ mod \ p$

ECDSA

- Same thing as DSA, just EC
- Public parameters: $(Curve, G, n)$
- Keys: $(x, Q = x \cdot G)$

Signing

- Generate "nonce" $k \in \{1, n-1\}$
- $R(r_x, r_y) = k \cdot G$
- $r = r_x \ mod \ n$
- $s = \frac{H(m) + xr}{k} \ mod \ n$
- Output (r, s)

Verification

- Check $r, s \neq 0, Q \in E(\mathbb{F}), Q \neq 0$
- $u_1 = \frac{H(m)}{s} \; mod \; n$
- $u_2 = \frac{r}{s} \; mod \; n$
- $R_{new}(r_x, r_y) = u_1 \cdot G + u_2 \cdot Q$
- Check $r_x \neq 0, r = r_x \; mod \; n$

Correctness

$$u_1 \cdot G + u_2 \cdot Q = \frac{H(m)}{s} \cdot G + \frac{r}{s} \cdot x \cdot G =$$

$$= \frac{H(m) + rx}{s} \cdot G = k \cdot G = R$$

What should stay secret in DSA/ECDSA?

- Private key x
- "Nonce" k

For a known k: $x = \frac{sk - H(m)}{r}$

DSA/ECDSA issues

- Nonce reuse
- Biased nonce
- Timing leak

Nonce reuse

$$s' = \frac{H(m') + rx}{k}$$

$$s'' = \frac{H(m'') + rx}{k}$$

Nonce reuse

$$s''(H(m') + rx) = s'(H(m'') + rx)$$

$$x = \frac{s''H(m') - s'H(m'')}{r(s' - s'')}$$

Biased nonce

- Known high or low bits of k can lead disclosure of x
- Need lots of signatures (depends on the bias)
- Use Lattice methods

Timing leak

- When computing $R = k \cdot G$ or $R = g^k$
- We can pick signature with fastest times and send them to biased nonce solver

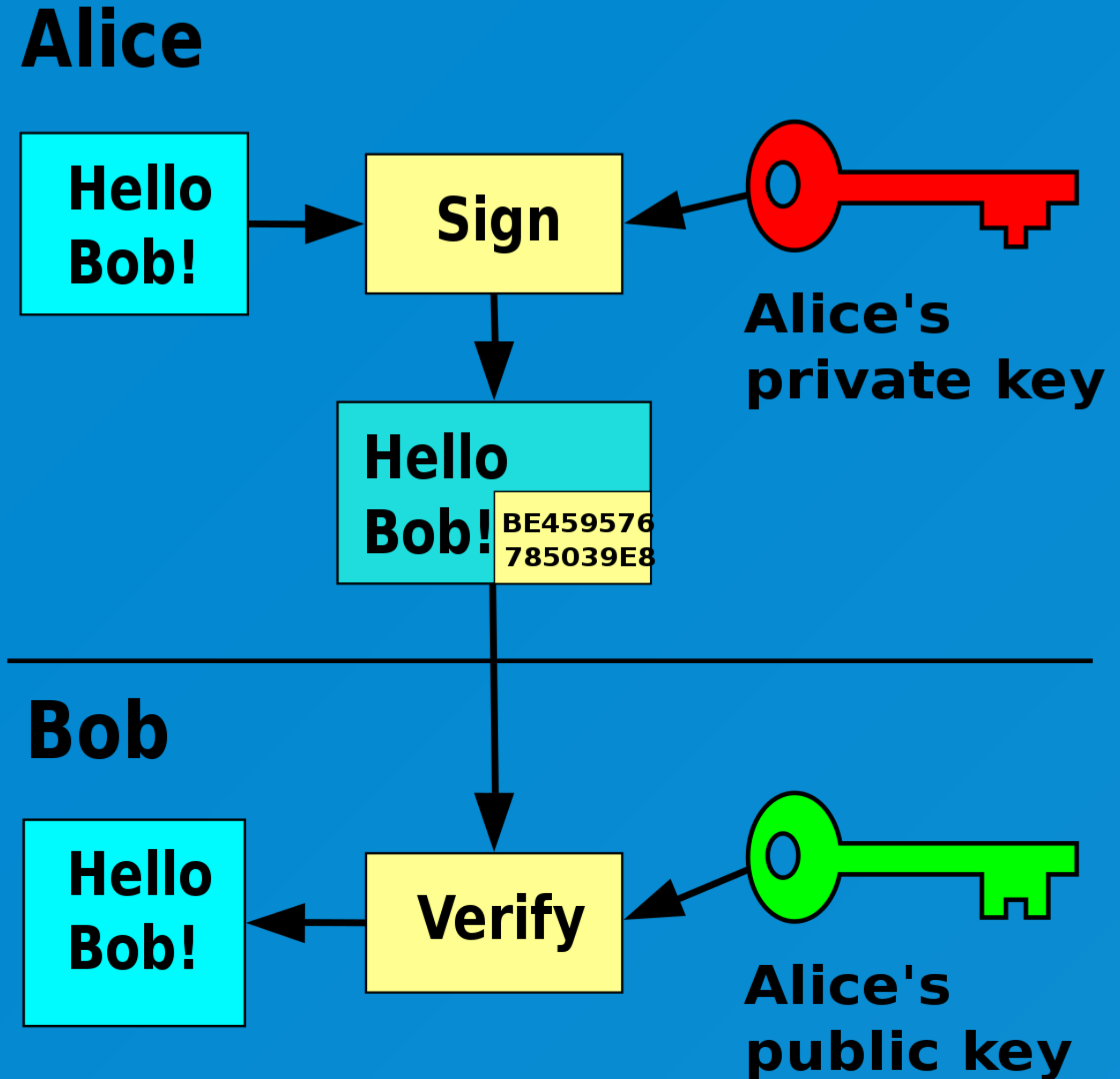
Find the bug

```
#include <ctime>
#include <thread>
#include <string>
struct  Signature {
    bigint r;
    bigint s;
};
Signature SafeECDSA(std::string hash){
    std::clock_t c_start = std::clock();
    Signature rs = LeakyECDSA(hash);
    std::clock_t c_end = std::clock();
    std::this_thread::sleep_for(
        std::chrono::microseconds(1000-1000000*(c_end-c_start)/CLOCKS_PER_SEC));
    return rs;
}
```

Constructing Multisigs for Traditional Signatures

Why do we sign?

We authenticate data (transactions, commands, logs, etc)



Why do we need multisignatures at all?

- Regular signatures need the key in plaintext
- Even in tokens it is possible to steal the key
- The purpose of multisignatures is to create a signature without exposing the key

Is it hard constructing Multiparty RSA/ECDSA?

- Everyone can be an adversary
- You have to check each communication round for correctness
- You have to give 0 chance to leak your private share

Insecure example (Avalanche Signatures)

- Two parties: P_1, P_2
- Private key $x = x_1 + x_2$
- Public key $X = X_1 + X_2 = x_1 \cdot G + x_2 \cdot G$

Please pay close attention to the protocol

$P_{0,1}$ breaks the protocol by obtaining either $x_{1,0}$ or $k_{1,0}$

P_1	P_2
$k_1 \xleftarrow{R} \mathbb{Z}_q^*$ $K_1 = \frac{1}{k_1} \cdot G$ $K_1 \longrightarrow$ $\Pi^{\mathrm{SHR}}(K_1; 1/k_1) \longrightarrow$	Check $\Pi^{\mathrm{SHR}}(K_1)$. $k_2 \xleftarrow{R} \mathbb{Z}_q^*, K_2 = \frac{1}{k_2} \cdot G$ $R = \frac{1}{k_2} \cdot K_1$ $r = x(R)$ $\alpha_{\bullet 22} = k_2(m + r x_2)$ $K_2 \longrightarrow$ $\Pi^{\mathrm{SHR}}(K_2; 1/k_2) \longrightarrow$ $\alpha_{\bullet 22} \longrightarrow$

P_2	P_1
Check $\Pi^{\text{SHR}}(K_1)$. $k_2 \xleftarrow{R} \mathbb{Z}_q^*, K_2 = \frac{1}{k_2} \cdot G$ $R = \frac{1}{k_2} \cdot K_1$ $r = x(R)$ $\alpha_{\bullet 22} = k_2(m + rx_2)$ $K_2 \longrightarrow$ $\Pi^{\text{SHR}}(K_2; 1/k_2) \longrightarrow$ $\alpha_{\bullet 22} \longrightarrow$	Check $\Pi^{\text{SHR}}(K_2)$. $R = \frac{1}{k_1} \cdot K_2$ $r = x(R)$ $\alpha_{\bullet 22} \cdot K_2 \stackrel{?}{=} m \cdot G + r \cdot X_2$ $\alpha_{1\bullet 1} = k_1(m + rx_1)$ $\alpha_{122} = k_1 \alpha_{\bullet 22}$ $\alpha_{1\bullet 1}, \alpha_{122} \longrightarrow$

P_1	P_2
Check $\Pi^{\mathrm{SHR}}(K_2)$. $R = \frac{1}{k_1} \cdot K_2$ $r = x(R)$ $\alpha_{\bullet 22} \cdot K_2 \stackrel{?}{=} m \cdot G + r \cdot X_2$ $\alpha_{1 \bullet 1} = k_1(m + rx_1)$ $\alpha_{122} = k_1 \alpha_{\bullet 22}$ $\alpha_{1 \bullet 1}, \alpha_{122} \longrightarrow$	$\alpha_{1 \bullet 1} \cdot K_1 \stackrel{?}{=} m \cdot G + r \cdot X_1$ $\alpha_{122} \cdot R \stackrel{?}{=} m \cdot G + r \cdot X_2$ $\alpha_{121} = k_2 \alpha_{1 \bullet 1}$ $\alpha_{121} \longrightarrow$

P_2	P_1
$\alpha_{1\bullet 1} \cdot K_1 \stackrel{?}{=} m \cdot G + r \cdot X_1$ $\alpha_{122} \cdot R \stackrel{?}{=} m \cdot G + r \cdot X_2$ $\alpha_{121} = k_2 \alpha_{1\bullet 1}$ $\alpha_{121} \longrightarrow$	$\alpha_{121} \cdot R \stackrel{?}{=} m \cdot G + r \cdot X_1$

Insecure example (Avalanche Signatures)

If $m = \frac{H(m)}{2}$, then

$$\alpha_{121} + \alpha_{122} = k_1 k_2 (2m + r(x_1 + x_2)) = k_1 k_2 (H + r(x_1 + x_2))$$

$$r = (\frac{1}{k_1 k_2} \cdot G).x$$

(In this case $k = \frac{1}{k_1 k_2}$)

Some obvious problems

- If someone gives us a and we have value b, we can't just return $c = ab$ (need to obfuscate)
- To defend against active adversary we need to check correctness of each value

Secure Multiplication Problem

- Let's say there are three parties with x_0, x_1, x_2 and y_0, y_1, y_3
- We want to compute $xy = (x_0 + x_1 + x_2)(y_0 + y_1 + y_2)$
- Everyone can compute $x_i y_i$
- How do we compute $x_i y_j, i \neq j$?
- Several ways, let's look at Homomorphic Encryption

Pallier cryptosystem

- As in RSA pick primes p and q ($n = pq$): $\gcd(pq, (p-1)(q-1)) = 1$
- $\lambda = \operatorname{lcm}(p-1, q-1)$
- $g \in \mathbb{Z}^*_{n^2}$
- $L(x) = \frac{x-1}{n}$
- $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$
- Public key (n, g)
- Private key (λ, μ)

Encryption/Decryption

- Check message $m \in 1, n - 1$
- Pick random $r \in 1, n - 1$
- $c = \text{Enc}(m) = g^m r^n \ mod \ n^2$
- $m = \text{Dec}(c) = L(c^\lambda \ mod \ n) \cdot \mu \ mod \ n$

Homomorphism

- $c_1 \cdot c_2 = g^{m_1+m_2}(r_1 r_2)^n \ mod \ n^2$
- $c^k = g^{mk}(r^k)^n \ mod \ n^2$

Multiplication to Addition with Pallier

1. P_i generates key
2. P_i encrypts x_i with Pallier -> c_i and sends to P_j
3. P_j samples $\beta_{ij} \in \{0, n-1\}$, computes $c_{ij} = c_i^{y_j} \cdot \mathrm{Enc}(\beta_{ij})$, sends c_{ij} to P_i
4. P_i decrypts c_{ij}, gets $x_i \cdot y_j + \beta_{ij}$

MtA with Pallier

- P_i holds $a_{ij} = x_i \cdot y_j + \beta_{ij}$
- P_j holds $b_{ij} = \beta_{ij}$

MtA with Pallier

- Each P_i of l parties communicates with each P_j, $j \neq i$
- P_i computes $z_i = x_i \cdot y_i + \sum_{j=0, j\neq i}^{l-1} a_{ij} - \sum_{j=0, j\neq i}^{l-1} b_{ji}$
- Each P_i publishes z_i

$$z = \sum_{i=0}^{l-1} z_i = \sum_{i=0}^{l-1}(x_i y_i) + \sum_{i,j=0, j\neq i}^{l-1}(x_i y_j + \beta_{ij}) - \sum_{i,j=0, j\neq i}^{l-1} \beta_{ij} = \sum_{i=0, j=0}^{l-1}(x_i y_j) = \sum_{i=0}^{l-1} x_i \sum_{i=0}^{l-1} y_j = xy$$

Other mechanisms

- MtA can be achieved with ElGamal (in the exponent), Oblivious Transfer, Class Groups
- Public computation of values a/b can be achieved by blinding both a and b with γ $(a\gamma, b\gamma)$
- Correctness is often achieved through Zero Knowledge Proofs

Multiparty construction problems RSA/ECDSA

- With ECDSA the curve itself already exists
- P_i needs to pick x_i and k_i during signing
- RSA requires you to construct N

RSA, constructing N

Two approaches:

1. Each P_i picks $p_i, q_i, N = \prod_i (p_i q_i)$
   Number of parties is limited (otherwise primes become way too small)
2. $N = (p_0 + ... + p_{l-1})(q_0 + ... + q_{l-1})$ and we have to check bipramility for each N

So RSA is somewhat painful

Aggregate signatures

Pairing

- G_1, G_2, G_T (G_1 and G_2 are curves), such that $|G_1| = |G_2| = |G_T| = p$
- $g_1 \in G_1, g_2 \in G_2, g \in G_T$
- A pairing function $e(a \cdot g_1, b \cdot g_2) = g^{(ab)}$

Boneh-Lynn-Shacham (BLS)

- Pick secret key $x \in \{1, p-1\}$
- Public key $X = x \cdot g_2$
- $s = \text{Sign}(m, x) = x \cdot H(m), H(m) \in G_1$
- $\text{Verify}(s, m) : e(s, g_2) = e(H(m), X)$

BLS

$$e(s, g_2) = e(x \cdot H(m), g_2) = e(H(m), g_2)^x =$$
$$= e(H(m), x \cdot g_2) = e(H(m), X)$$

Cool BLS feature

- $X_i = x_i \cdot g_2$
- $s_i = \text{Sign}(m_i, x_i)$
- $s = \sum_i s_i$
- $e(s, g_2) = \prod_i (e(H(m_i), X))$

Special case

- $\forall i \; m_i = m$
- $e(s, g_2) = e(H(m_i, \sum_i (X_i)))$

Threshold Signatures

- We can use Shamir's Secret Sharing Scheme
- (t, n) secret sharing

Shamir's Secret Sharing Scheme

- Secret a
- n
- $t + 1$ should be able to restore a
- t shouldn't

Shamir's Secret Sharing Scheme

- Construct polynomial $f(x) = a_0 + ...a_1x + ...a_tx^t$
- $a_0 = a$
- Give each party P_i $f(i)$
- We can always reconstruct from $t + 1$ through lagrange interpolation
- t is not enough

Distributed Key Generation

- We can construct a so that noone knows it
- Each P_i creates their own f_i
- Sends each P_j $f_i(j)$
- Publishes $A_{i0} = a_{i0} \cdot g_2, \ldots A_{it} = a_{it} \cdot g_2$
- Each P_j computes $F_{ij} = \sum_{k=0}^{l-1} A_{ik} j^k$
- Checks $e(f_{ij}, g_2) = e(g_1, F_{ij})$
- Each P_i's key is $\sum_{j=0}^{l-1} f_j(i)$
- Common public key is $\sum_{i=0}^{l-1} A_{i0}$

Attacks on Aggregate Signatures

- $x_1 + x_2 = 0, X_1 = x_1 \cdot g_2, X_2 = x_2 \cdot g_2$
- $s_1 = x_1 \cdot H(m), s_2 = x_2 \cdot H(m)$
- $e(s_1 + s_2, g_2) = e(g_1, X_1 + X_2)$
- What's the problem?

Attacks on Aggregate Signatures

- Some party has public key X_1
- Generate $X_2 = x_2 \cdot g_2$
- Publish $X_2' = X_2 - X_1$ as your public key
- Aggregate key $X = X_1 + X_2' = X_2$

Attack on DKG

- G_2 is usually part of a larger group E
- During DKG attacker can submit points on E instead of G_2
- Under certain condition the attacker can poison the public key, making it unusable

Thank you for your attention

I'd like:

- Jean-Philippe Aumasson
- Adrian Hamelink
- Omer Shlomovits
- Nguyen Thoi Minh Quan

for their research.